

B E S C H R E I B U N G

des

A S S E M B L E R P A K E T E S

von

Gregor Lohmann

A S S E M B L E R P A K E T

Lieber TI-Freund!

Mit diesem Assemblerpaket erhalten Sie eine kostengünstige Alternative zum Mini-Memory Modul, bzw. zum Editor/Assembler von Texas Instruments. Mit Hilfe der rechts an der Konsole anschließbaren Schaltung und dem Softwarepaket ist ein kompletter Line-by-Line Assembler und ein Einbinden von Assemblerprogrammen in Extended-Basic möglich. Voraussetzungen sind lediglich die TI 99/4A Konsole und das Extended-Basic Modul.

Inbetriebnahme:

Die Hardwareschaltung wird am Systembus rechts an die Konsole angesteckt. Der Schiebeschalter wird in die linke Position geschoben. Die Spannungsversorgung wird angestellt und mit 2 Extended-Basic aufgerufen. Der Schiebeschalter wird nun wieder in die entgegengesetzte Position gebracht und der Befehl CALL INIT eingegeben. Nach Eingabe von NEW ist der Rechner - wie bisher in Extended-Basic - betriebsbereit. Jedoch ist zusätzlich der Befehl CALL LOAD freigegeben.

Software:

Folgende Software wird mit der Kassette geliefert:

1.) Assembler
2.) Disassembler
3.) Kassettenrecorder-Abspeicherroutine
4.) Kassettenrecorder-Einleseroutine
5.) Monitorprogramm, davor die Daten der benutzten Assemblerprogramme

Auf der 2.Seite:

1.) Erklärungen
2.) Daten zu einem Demonstrationsprogramm

.....

B E S C H R E I B U N G

des

A S S E M B L E R P A K E T E S

von

Gregor Lohmann

ausgelegt für den TI99/4A in Verbindung mit Konsole, Kassetten-recorder und Extended-Basic. Bestehend aus einem Softwarepaket und einer an der Konsole anschließbaren Hardwareschaltung

Vertrieb: Firma Radix Bürotechnik, Bornstr.4, 2000 Hamburg 13

zu 1.): Das Assembler-Programm stellt einen komfortablen Line-by-Line Assembler zur Erstellung von Maschinenprogrammen im Rambereich von 24F4 - 27FF dar. Die Möglichkeiten des Programms sind umfangreicher als die des Minimem Moduls.
Nach dem Programmstart meldet sich das Programm mit der niedrigsten verfügbaren Programmadresse (24F4). Die Eingaben erfolgen nach folgendem Muster: "(Label) Opcode Operandfeld", wobei die runden Klammern optionale und die nicht eingeklammerten notwendige Eingaben kennzeichnen. Wird kein Label gesetzt, so ist ein Leerzeichen einzugeben. Die Opcodes und Operanden werden durch ein Leerzeichen getrennnt. Kommentare sind nicht zulässig!
Ein Label darf die Länge von 25 Zeichen nicht überschreiten.
Das gleiche gilt auch für Definitionen. Die Anzahl der Vorwärtsreferenzen ist auf 15 begrenzt. Vorweggenommene Definitionen, wie beim Editor/Assembler, gibt es nicht. Es wird - genauso wie beim Minimem Modul - mit "Label EQU Adresse" definiert.
Da wir einen Line-by-Line Assembler haben, werden die Programme mit Absolut-Adressen eingegeben. Sie sind also nicht auf andere Speicherstellen verschiebbar. Der Befehl RORG fehlt. Bei jedem DATA-Befehl kann nur eine Zahl eingegeben werden. Ein Editieren des vorher eingegebenen Textes ist nicht möglich; jedoch kann die Eingabe von einem Drucker dokumentiert werden. Die Ausdruckroutine ist für den Drucker GP 700A (Anschluß über parallele Schnittstelle) ausgelegt. Ein Neuaufrollen des Maschinencodes ist aber mit dem Befehl AORG möglich.

Folgende Assembler Befehle werden vom Programm verarbeitet:

Arithmetische Befehle:

A	ADD WORDS	addiert eine Kopie des Source-Operanden zum zum Destination-Operanden, Ergebnis: Source + Destination = Destination
AB	ADD BYTES	addiert eine Kopie des höherwertigen Bytes der Source zum höherwertigen Byte der Destination

.....

ABS	ABSOLUTE VALUE	ersetzt den Wert der Source mit ihrem Absolutwert
AI	ADD IMMEDIATE	addiert einen unmittelbaren Operanden zum Workspace-Register
DEC	DECREMENT	vermindert Source um 1 und ersetzt Source mit dem Resultat
DECT	DECREMENT BY TWO	vermindert Source um 2 und ersetzt Source mit dem Resultat
DIV	DIVIDE	divi diert den Inhalt zweier aufeinanderfolgender Register durch den Wert eines Source-Operanden und placiert den Quotienten im ersten, den Rest im zweiten der aufeinanderfolgenden Register
INC	INCREMENT	erhöht Source um 1 und setzt Ergebnis in Source
INCT	INCREMENT BY TWO	erhöht Source um 2 und setzt Ergebnis in Soource
MPY	MULTIPLY	multipliziert mit dem Source-Operanden ein Register und plaziert das 32-bit Ergebnis ab einem Register vor dem ursprünglichen
NEG	NEGATE	tauscht Source gegen ihren negativen Wert aus
S	SUBSTRACT WORDS	substrahiert die Source von der Destination und tauscht diese gegen das Ergebnis aus
SB	SUBSTRACT BYTE	wie S nur auf das höherwertige Byte bezogen

<u>Lade- und Kopierinstruktionen:</u>

LI	LOAD IMMEDIATE	übernimmt den unmittelbaren folgenden Operanden in das angesprochene Workspace-Register
LIMI	LOAD INTER-RUPT MASK	übernimmt Bit 12-15 des unmittelbar folgenden Wortes in die Interruptmaske des Status-Registers
LWPI	LOAD WORKSPACE POINTER IMME-DIATE	ersetzt den Inhalt des WP mit dem nächst-folgenden Wort
MOV	MOVE WORDS	ersetzt den Destinationoperand mit einer Kopie des Sourceoperanden ohne diesen zu verändern
MOVB	MOVE BYTE	wie MOVE, nur auf das höherwertige Byte bezogen

.....

STST	STORE STATUS	speichert den Inhalt des Stausregisters in einem Workspace Register
STWP	STORE WORKSPACE	speichert WP in einem Workspace Register
SWPB	SWAP BYTES	vertauscht Bits 0-7 mit Bits 8-15 des Sourcce-Operanden

<u>Logische Instruktionen:</u>

ANDI	AND IMMEDIATE	unterzieht den Source Operanden einer bitweisen AND-Operation mit dem unmittelbar folgenden Wort
ORI	OR IMMEDIATE	wie ANDI, nur OR-Operation
XOR	EXCLUSIVE OR	unterzieht Source und Destination einer bit-weisen XOR-Operation
INV	INVERT	ersetzt den Source-Operanden mit seinem Einer-komplement
CLR	CLEAR	setzt den Source-Operand auf 16-Bit "0"
SETO	SET TO ONE	setzt den Source-Operand auf 16 Bit "1"
SOC	SET ONES CORRESPONDING	Source und Destination werden mit OR ver-bunden. Ergebnis in der Destination
SOCB	SET ONES CORRESPONDING BYTE	wie SOC nur auf das höherwertige Byte bezogen
SZC	SET ZEROS CORRESPONDING	AND Operation zwischen Source und Destination ersetzt Destination durch das Resultat
SZCB	SET ZEROS CORRESPONDING BYTE	wie SZC, nur auf das höherwertige Byte bezogen

<u>Shift-Instruktionen:</u>

SRA	SHIFT RIGHT ARITHMETIC	schiebt den Registerinhalt um die anzugebende Bitzahl nach rechts, leere Positionen werden mit dem Vorzeichenbit aufgefüllt
SRL	SHIFT RIGHT LOGICAL	wie SRA, nur Auffüllung mit "0"
SLA	SHIFT LEFT ARITHMETIC	schieben nach links mit Auffüllung von "0"

.....

SRC	SHIFT RIGHT CIRCULAR	schiebt nach rechts und bringt die herausgeschobenen Bits links wieder herein

Pseudoinstruktionen:

NOP	NO OPERATING	ergibt den Maschinencode 1000, entspricht einem Sprung zur nächsten Adresse
RT	RETURN	ergibt den Maschinencode 045B wie B *R11

Sprünge und Verzweigungen:

B	BRANCH	wechselt den Inhalt des Programmzählers gegen die Sourceadresse aus
BL	BRANCH AND LINK	wie oben und speichert die unmittelbar folgende Adresse als Rücksprungadresse in R11
BLWP	BRANCH AND LOAD WORKSPACE POINTER	tauscht den Workspace-Pointer gegen den Sourceoperanden und den PC gegen das unmittelbar dem Sourceoperanden folgende Wort aus; im neuen Workspace werden die alten Werte gespeichert: R13=WP, R14=PC, R15=STATUS
JEQ	JUMP IF EQUAL	Sprung bei gesetztem Equal-Bit
JGT	JUMP IF GREATER THAN	Sprung bei gesetztem "Arithmetic greater-than Status"-Bit
JHE	JUMP IF HIGH OR EQUAL	Sprung bei gesetztem Equal und Logisch-High-Bit
JH	JUMP IF LOGICAL HIGH	Sprung bei gesetztem Logisch-größer-Bit
JL	JUMP IF LOGICAL LOW	Sprung bei nicht gesetztem Equal- und Logisch-High-Bit
JLE	JUMP IF LOW OR EQUAL	Sprung bei entweder gesetztem Equal- oder nicht gesetztem Logisch-größer-Bit
JLT	JUMP IF LESS THAN	Sprung bei nicht gesetztem Equal und Arithmetisch-größer-Bit
JMP	UNCONDITIONAL JUMP	unbedingter Sprung zur nachfolgenden Adresse
JNC	JUMP IF NO CARRY	Sprung bei nicht gesetztem Carry-Bit
JNE	JUMP IF NOT EQUAL	Sprung bei nicht gesetztem Equal-Bit

JNO	JUMP IF NO OVERFLOW	Sprung bei nicht gesetztem Overflow-Bit
JOP	JUMP IF ODD PARITY	Sprung bei gesetztem Odd-Parity-Bit
JOC	JUMP ON CARRY	Sprung bei gesetztem Carry-Bit
RTWP	RETURN WITH WORKSPACE POINTER	Umkehrung des BLWP-Befehls
X	EXECUTE	verarbeitet den Source-Operanden als Instruktion
XOP	EXTENDED OPERATING	benötigt Transfer-Vektoren in zwei aufeinanderfolgenden Speicherstellen (WP,PC), auf die der Operand zeigt. im XOP-Workspace werden die Rücksprunginformationen wie bei BLWP gespeichert. Nicht auf allen TI 99 möglich !

Vergleichsoperationen:

C	COMPARE WORDS	vergleicht den Source-Operanden mit dem Destinationswort und setzt/löscht entsprechende Status-Bits
CB	COMPARE BYTES	wie C, nur auf die höherwertigen Bytes der Worte bezogen
COC	COMPARE ONES CORRESPONDING	setzt Equal-Bit bei Übereinstimmung der logisch-Eins-Bits der Source mit denjenigen der Destination
CZC	COMPARE ZERO CORRESPONDING	wie COC, nur auf logisch Null bezogen

Kontroll- und CRU-Instruktionen:

LDCR	LOAD CRU	überträgt eine festgelegte Bitanzahl in die CRU, die CRU-Adresse ist dabei in Bit 3 bis 14 von R12 enthalten
SBO	SET CRU BIT TO ONE	setzt ein Bit der CRU auf 1, relativ zur CRU-Adresse in R12, ohne den Inhalt zu verändern
SBZ	SET CRU BIT TO ZERO	wie SBO nur setzen auf "0"
STCR	STORE CRU	überträgt die angegebene Bitanzahl aus der CRU in eine anzugebende Speicherzelle
TB	TEST BIT	vergleicht Einzelbit der CRU und setzt/löscht Equalbit

Wie man sieht, unterstützt der Extended-Basic Assembler den kompletten lauffähigen Befehlssatz des TMS9900-Prozessors.
Ebenso werden sämtliche Adressierungsarten unterstützt:

Workspace register adressing:	z.B. MOV 4,8 kopiert den Inhalt von Register 4 in R8 - der Extended-Basic Assembler akzeptiert Register ohne vorangestelltes R und nur als Dezimalzahl!
Workspace register indirect:	z.B. A *7,*2 bedeutet: addiere das Wort an der Adresse des Inhalts von R7 zum Wort an der Adresse des Inhalts von R2
Symbolische Adressen:	z.B. C αADR,αLABEL bedeutet: Vergleiche den Inhalt der Worte auf den durch die Label "ADR" und "LABEL" definierten Adressen.
Indexed memory adressing:	z.B. Sα2(7),αLIST(5) bedeutet: substrahiere den Inhalt jener Adresse, die die Summe der Registerinhalte R2,R7 darstellt, vom Wort an der Stelle LIST + Inhalt von R5
Workspace register indirect autoincrement adressing:	z.B. C *3+,2 bedeutet: Vergleiche den Inhalt an jener Adresse, auf die R3 zeigt mit Registerinhalt R2 und erhöhe R3 um zwei (bei Byteoperationen wird nur um 1 erhöht).

Mit folgenden Befehlen läßt sich das Extended-Basic Assemblerprogramm steuern:

AORG	setzt die Programmadresse auf den gewünschten Wert. So kann man auch den Bereich von 2000-24F4 erreichen. Doch Vorsicht, es stehen dann nicht mehr alle Hilfsprogramme zur Verfügung.
BSS	Blockstartsymbol, reserviert den gewünschten Speicherraum. Die Reservierungsangabe kann Hexadezimal oder dezimal erfolgen. Dies gilt übrigens für fast alle Befehle.
DATA	schreibt den angegebenen Wert an die aufgelistete Adresse (Nur ein Wort pro DATA ist zulässig)
TEXT	schreibt die angegebene String-Konstante in den Speicher, wird gegebenenfalls auf eine geradzahlige Wortlänge mit Blank ergänzt.
EQU	definiert konstante Adressen

END	schließt das Programm ab und leitet die Namesroutine ein.
LABLIST	listet alle bisherigen Label mit der zugehörigen Adresse auf
REFLIST	listet alle definierten Referenzen auf.
VORREF	listet alle derzeit offenen Vorwärtsreferenzen auf
DIS	löscht das Assemblerprogramm und verknüpft zu dem auf der Kassette nachfolgenden Disassembler.

Zu der END-Instruktion:
Um den Programmzähler nicht umstellen zu müssen, wird mit der END-Instruktion eine bequeme Benennungsroutine eingeleitet. Programme werden nur dann angenommen, wenn die Anfangsadresse im Bereich 2000-27FF liegt und auf 2004 der Beginn der Namenstabelle steht. Die Routine fragt zuerst nach dem Programmnamen, der mit max. 6 Zeichen angegeben werden kann. Danach wird die Programmstartadresse abgefragt, die in Form eines Labels eingegeben werden sollte.
Nach dem ersten Programmnamen können beliebig weitere eingegeben werden, falls man mehrere Assemblerprogramme auf einmal geschrieben hat.
Nach Beenden dieser Eingaben wird das Programm gestoppt und die Maschinenprogramme können als Teil eines Extended-Basic Programms oder direkt mit CALL LINK aufgerufen werden.

Zusammenfassung:
Es werden vom Extended-Basic Assembler alle Optionen des Editor/Assemblers unterstützt mit folgenden Ausnahmen:

1.) Nur eine Zahl pro DATA-Anweisung
2.) Außer Indizierung keine Berechnung im Operandenfeld
3.) Die Programme sind auf eine bestimmte Adresse festgelegt
4.) Definitionen sind nicht vorgegeben
5.) Das Symbol $ als Synonym für den Wert des Programmzählers ist nicht verwendbar.

Zu 2.): Disassembler:

Will man sich den Assemblertext zu einmal erstellten Programmen ansehen, so bietet diese Programm eine wertvolle Unterstützung. Nach Eingabe der zu untersuchenden Speicherstellen wird der Assemblertext auf dem Bildschirm oder wahlweise auf einem Drucker (parallele Schnittstelle) aufgelistet. Man kann ebenfalls das Betriebssystem und unbekannte Maschinenprogramme untersuchen. Der gesamte Speicherbereich des Rechners ist mit diesem Programm ansprechbar (Alle RAMS und ROMS).
Das Listen kann durch Drücken der Leertaste angehalten werden, bei Betätigung der ENTER-Taste erscheint das Anfangsmenü zur Eingabe einer neuen Startadresse. Die Tasten müssen dabei längere Zeit gedrückt gehalten werden.

Zu 3.): Kassettenrecorder Abspeicherroutine:

Damit einmal erstellte Assemblerprogramme nicht bei Abschalten des Rechners gelöscht werden, haben Sie mit diesem Programm die Möglichkeit, den Maschinencode in kürzester Zeit auf Kassette zu sichern. Es sind keine komplizierten Eingaben nötig. Die Namenstabelle, Programmstartadresse und Länge der Programme sucht sich dieses Programm selbst. Auf Wunsch ist es selbstverständlich möglich, einen speziellen Speicherbereich selbst zu wählen.
Dazu halten Sie die Tasten FCTN REDO gedrückt. Nach einiger Zeit werden Sie dann nach den neuen Anfangs- und Endadressen gefragt, die dezimal einzugeben sind.

Zu 4.): Kassettenrecorder-Einleseroutine:

Die unter Punkt 3 beschriebenen abgespeicherten Software-Programme lassen sich mit dieser Routine wieder einlesen.
Werden in einem Extended-Basic Programm Assemblerroutinen benötigt, so ist vor Einladen des Extended-Basic Programms mit dieser Routine ein Einlesen der Assembler-Programme nötig. Selbstverständlich sind die Assemblerprogramme mit CALL LINK auch im Editiermodus aufrufbar.

.....

Zu 5.): Monitor-Programm:

Diese Software besteht z.B. aus einer Kombination von Assembler und Extended-Basic Programmen. Zuerst werden mit Hilfe der Einleseroutine die Assemblerhilfsprogramme eingelesen und dann das eigentliche Monitor-Programm.

Es bietet folgende Möglichkeiten:

1.) Ein Auslesen des VDP-Rams im ASCII-Format
2.) Ein Auslesen der graphischen Roms
3.) Manipulation des VDP-Rams

Als Hilfe bietet es noch die Umrechnung von hexadezimalen Zahlen in dezimale und umgekehrt. Mit den außerdem nun möglichen Befehlen CALL LOAD und CALL PEEK kann nun jeder Speicherplatz untersucht und gegebenenfalls geändert werden.

Um die Möglichkeiten dieses Softwarepakets nutzen zu können, sind folgende Informationen über die Nutzung des Speicherplatzes notwendig. In der mitgelieferten Hardwareschaltung befindet sich ein RAM, der den Bereich von 2000-27FF benutzt. Diese Speicherbelegung ist vom Betriebssystem vorgesehen:

Benutzung des Erweiterungs-RAM vom Extended-Basic

```
2000  +----------------------------------------------------+
      ! XML Vektor benutzt vom Interpreter     (205A)      !
2002  +----------------------------------------------------+
      ! Vektor zur ersten freien Speicheradresse           !
2004  +----------------------------------------------------+
      ! Vektor zum Beginn der DEF-Liste                    !
2006  +----------------------------------------------------+
      ! Identifizierungscode AA55                          !
2008  +----------------------------------------------------+
      ! Hilfsprogramm BLWP Vektoren                        !
2038  +----------------------------------------------------+
      ! Workspace für die Hilfsroutinen                    !
2058  +----------------------------------------------------+
24F4  ! Hilfsroutinen                                      !
      +----------------------------------------------------+
      ! Eigene Assembler-Programme                         !
27xx  +----------------------------------------------------+
      ! Def-Liste                                          !
27FF  +----------------------------------------------------+
```

.....

Benutzung des CPU-Rams

Der in der Computerconsole eingebaute RAM, genannt der "CPU-Ram", benutzt den Adressbereich von 8300 - 83FF. Dieser Bereich wird in Abhängigkeit von den laufenden Programmen wie folgt benutzt:

8300-8349 Dieser Bereich kann für Ihre eigenen Programme mit folgenden Einschränkungen genutzt werden. Die Systemsoftware benutzt diesen Teil nur als Zwischenspeicher für Aufrufe von CALL LINK oder CALL LOAD aus dem Basic.

834A-836D Genutzt vom Betriebssystem für den Stack, Gleitkommaberechnungen und DSR-Routinen. Falls keine Hilfsroutinen des Betriebssystems im Assemblerprogramm benutzt werden, steht dieser Bereich voll zur Verfügung.

836E-837F Speicher für den GPL Status-Block:

8370 Enthält die höchste verfügbare Adresse im VDP-Ram.

8372 Das niederwertige Byte des Daten-Stack Zeigers. Das höherwertige Byte hat den Wert 83. Nach dem Initialisieren des Computers steht hier ein Wert von 9F, nach der ersten Benutzung A0.

8373 Das niederwertige Byte des Unterprogramm-Stack Zeigers. Das Höherwertige ist ebenfalls immer 83. Am Anfang steht hier der Wert 7E, nach dem ersten Aufruf der Wert 80.

8374 Die Tastaturzahl nach Abfrage der Tastatur.

8375 Der bei Abfrage der Tastatur entdeckte ASCII-Code.

8376 Der Y-Wert der Fernsteuerung

8377 Der X-Wert der Fernsteuerung

8378 Der Zufallszahlengenerator

8379 VDP - Interrupt Zähler. Er wirde alle sechzigstel Sekunde um 1 erhöht (Ideal zum Aufbau einer Software Uhr).

837A Die Anzahl der Sprites, die in Bewegung sind. Normal auf dem Wert 0. Mit CALL LOAD auf diese Adresse können so alle bewegten Sprites abrupt gestoppt und wieder in Bewegung versetzt werden.

.....

837B Das VDP-Status Byte. Es ist eine Kopie des VDP-Registers.

Bit 0 ist der 60 Hz VDP-Interrupt
Bit 1 gibt an, ob mehr als fünf Sprites in einer Zeile sind.
Bit 2 zeigt an, wenn sich zwei Sprites überlappen.
Bit 3-7 enthält die Nummer des fünften Sprites, falls fünf oder mehr Sprites in einer Zeile stehen.

837C Das GPL-Status Byte

Bit 0 ist das "High" Bit
Bit 1 ist das "Greater Than" Bit
Bit 2 ist das "Condition" Bit. Es wird außerdem von der Tastaturabfrage auf 1 gesetzt, falls eine neue Taste erkannt worden ist. Ebenso von den DSR-Routinen, um zu erkennen, ob ein File existiert.
Bit 3 ist das "Carry" Bit.
Bit 4 ist das "Overflow" Bit

837D Zeichenpuffer für den VDP-Ram. Er enthält den Wert, der auf dem Bildschirm neu dargestellt wird. Wird ein Wert auf diese Adresse geladen, so führt das zu einer Darstellung des entsprechenden Buchstabens auf dem Bildschirm an der Adresse, die in den folgenden Bytes steht.

837E Zeiger auf die aktuelle Bildschirmzeile.

837F Zeiger auf die zugehörige Spalte.

8380-83BF Dieser Bereich wird als Stack für die Unterprogramme und Daten genutzt. Solange man nicht die GPLLNK-Routine aufruft, kann man diesen Bereich nutzen. Vorraussetzung ist aber, daß man nach Gebrauch dieses Speichers die vorher darin enthaltenen Werte wieder hineinschreibt.

.....

83C0-83DF Hier liegt der Arbeitsbereich des Interpreters. Die Bytes sind nach folgendem Muster belegt:

83C0	Zufallszahl
83C2	Nur für TI99/4: 8 Bytes für interne Flags und die Remote-Steuerung Nur für TI99/4A: Flag-Byte für die Interrupt-Routine
83C4	Nur für TI99/4A: Adresse der vom Benutzer definierten Interrupt-Routine.
83CA	Daten zur Tastatur
83CC	Zeiger zur Sound-Liste
83CE	Zahl der Sound-Bytes, die nach jedem VDP-Interrupt um 1 erniedrigt werden.
83D0	Zeiger zum Ansprechen der GROMS und ROMS. Vier Bytes lang.
83D4	Aktueller Wert aus dem VDP-Register 1.
83D6	Zähler, um den Bildschirm abzuschalten. Wird nach jedem VDP-Interrupt um 1 erniedrigt und löscht den Bildschirm, wenn der Wert Null erreicht worden ist. Wird jedesmal nach Druck einer Taste wieder hochgesetzt.
83D8	Rücksprungadresse, gesichert von der Routine, die die Tastatur abfragt.
83DA	Nummer des Spielers am Joystick

83E0-83FF Hier stehen die Arbeitsregister des GPL (Graphik-Programming-Language) Interpreters. Dieser Bereich wird von allen Routinen in der Konsole benutzt; so auch vom Basic-Interpreter. Der genaue Gebrauch hängt jeweils von den arbeitenden Programmen ab. Unabhängig ist aber die Belegung der Register 13,14 und 15. Sie enthalten immer die GROM-Leseadresse, das Systemflag und die VDP-Schreibadresse.

Sämtliche Angaben sind ohne Gewähr. Sie beziehen sich allgemein auf das ganze System und nicht speziell auf den Betrieb mit Extended-Basic. Als Quelle diente das Editor/Assembler Handbuch. Bei Extended-Basic steht zum Beispiel bei 8386 die höchste freie Adresse der Speichererweiterung.

.....

Bei der Parameterübergabe mit CALL LINK an ein Assemblerprogamm wird der Bereich von 8300 - 8315 gebraucht.

Doch dazu folgendes:
Da die Hilfsroutinen zur Parameterübergabe für den Anschluß einer kompletten Speichererweiterung geschrieben worden sind, funktioniert die Übergabe nur für Strings. Diese werden nämlich auch bei Anschluß einer Speichererweiterung im VDP-Ram abgelegt.
Die numerischen Variablen sucht das Hilfsprogramm bei diesem Assemblerpaket vergeblich in der Speichererweiterung, weil sowohl das Programm, wie auch die Daten zum Programm im VDP-Speicher abgelegt sind. Daher erhält man den Wert 0 nach Aufruf des Hilfsprogramms.
Lösung des Problems: Übertragen Sie Parameter entweder immer als Strings , oder laden Sie die Übergabeparameter mit dem Befehl CALL LOAD in die von Ihnen definierten Register. Als Beispiel können Ihnen die in der Erklärung des Softwarepaketes aufgeführten Programme dienen.

Die Adressen der verschiedenen Unterprogramme des Betriebssystem, die Zeiger auf bestimmte Routinen, die Fehlermeldungen und andere wichtige Adressen sind von der benutzten Programmiersoftware zum Teil abhängig. Wenn man Programme, die für das Minimem-Modul oder den Editor/Assembler geschrieben worden sind, mit dem Extended-Basic laufen lassen will, so muß man einige Adressen ändern.
Folgende Zuweisungen gelten speziell für Extended-Basic:

VDPWA	EQU	8C02	VDPWD	EQU	8C00
VDPRD	EQU	8800	VDPSTA	EQU	8802
FAC	EQU	834A	GPLWS	EQU	83E0
PAD	EQU	8300	SOUND	EQU	8400
SPCHRD	EQU	9000	SPCHWT	EQU	9400
GRMRD	EQU	9800	GRMRA	EQU	9802
GRMWD	EQU	9C00	GRMWA	EQU	9C02
SCAN	EQU	000E			

.....

Und nun folgen die Verzeigungsvektoren der Hilfsprogramme, die im Speicherbereich von 2000 - 24F4 stehen:

NUMASG	EQU	2008	NUMREF	EQU	200C
STRASG	EQU	2010	STRREF	EQU	2014
XMLLNK	EQU	2018	KSCAN	EQU	201C
VSBW	EQU	2020	VMBW	EQU	2024
VSBR	EQU	2028	VMBR	EQU	202C
VWTR	EQU	2030	ERR	EQU	2034

Folgende Routinen stehen im System-Rom:

FADD	EQU	0D80	FSUB	EQU	0D7C
FMUL	EQU	0E88	FDIV	EQU	0FF4
SADD	EQU	0D84	SSUB	EQU	0D74
SMUL	EQU	0E8C	SDIV	EQU	0FF8
CSN	EQU	11AE	CFI	EQU	12B8
FCOMP	EQU	0D3A	NEXT	EQU	0070
COMPCT	EQU	00	GETSTR	EQU	02
MEMCHK	EQU	04	CNS	EQU	06
VPUSH	EQU	0E	VPOP	EQU	10
ASSGNV	EQU	18	CIF	EQU	20
SCROLL	EQU	26	VGWITE	EQU	34
GVWITE	EQU	36			

Die Fehlermeldungen haben folgende Equates:

ERRNO	EQU	0200	2	Numeric Overflow
ERRSYN	EQU	0300	3	Syntax Error
ERRIBS	EQU	0400	4	Illegal After Subprogramm
ERRNQS	EQU	0500	5	Unmatched Quotes
ERRNTL	EQU	0600	6	Name Too Long
ERRNM	EQU	0700	7	String-Number Mismatch
ERROBE	EQU	0800	8	Option Base Error
ERRMUV	EQU	0900	9	Improperly Used Name
ERRIM	EQU	0A00	10	Image Error
ERRMEM	EQU	0B00	11	Memory Full

.....

ERRSO	EQU	0C00	12	Stack Overflow
ERRNWF	EQU	0D00	13	NEXT Without FOR
ERRFNN	EQU	0E00	14	FOR-NEXT Nesting
ERRSNS	EQU	0F00	15	Must Be In Subprogramm
ERRRSC	EQU	1000	16	Recursive Subprogramm Call
ERRMS	EQU	1100	17	Missing SUBEND
ERRRWG	EQU	1200	18	RETURN Without GOSUB
ERRST	EQU	1300	19	String Truncated
ERRRBS	EQU	1400	20	Bad Subscript
ERRSSL	EQU	1500	21	Speech String too long
ERRLNF	EQU	1600	22	Line Not Found
ERRBLN	EQU	1700	23	Bad Line Number
ERRLTL	EQU	1800	24	Line Too Long
ERRCC	EQU	1900	25	Can't Continue
ERRCIP	EQU	1A00	26	Command Illegal In Programm
ERROLP	EQU	1B00	27	Only Legal in a Programm
ERRBA	EQU	1C00	28	Bad Argument
ERRNPP	EQU	1D00	29	No Programm Present
ERRBV	EQU	1E00	30	Bad Value
ERRIAL	EQU	1F00	31	Incorrect Argument in List
ERRINP	EQU	2000	32	Input Error
ERRDAT	EQU	2100	33	Data Error
ERRFE	EQU	2200	34	File Error
ERRIO	EQU	2400	36	I/O Error
ERRSNF	EQU	2500	37	Subprogramm Not Found
ERRPV	EQU	2700	39	Protection Violation
ERRIVN	EQU	2800	40	Unrecognized Character
WRNNO	EQU	2900	41	Numeric Overflow
WRNST	EQU	2A00	42	String Truncated
WRNPP	EQU	2B00	43	No Programm Present
WRNINP	EQU	2C00	44	Input Error
WRNIO	EQU	2D00	45	I/O Error

.....

ASSEMBLERPROGRAMMIERUNG

Mit diesem Kapitel können nur erste Hinweise zum Einstieg in das Programmieren mit Assembler gegeben werden. Zum Erlernen seien die angeführte Literatur, bzw. die auf dem Markt angebotenen Assemblerkurse empfohlen.

Mit diesem Assemblerpaket geschriebene Programme werden in Extended-Basic zum Starten mit dem Befehl CALL LINK entweder im Editier- oder im Programmmodus aufgerufen. Um nach Ablauf des Assemblerprogramms ordnungsgemäß zum Basic zurückzukommen, ist folgende Programmierung empfehlenswert.
Der Basic- und GPL-Interpreter haben ihre Register auf der Adresse 83E0 liegen. Für sein eigenes Programm benutzt man am besten eigene Register, um mit dem Basic nicht in Konflikt zu kommen. Diese eigenen Register müssen am Anfang des Programms geladen werden. Nach Ablauf des Programms werden dann wieder die alten Register des GPL-Interpreters geladen.
Ein solches Programm sieht dann so aus:

```
GPLWS EQU >83E0     Definition von GPLWS
MYWS BSS 32         Reserviert 32 Bytes für die eigenen Register
START LWPI MYWS     Mit Start fängt das eigentliche Programm an.
      .             Es werden die eigenen Register geladen.
      .
Eigener Programmtext
      .
      .
 LWPI GPLWS         Die alten Register werden wieder geladen.
 RT                 Rücksprung zum Extended-Basic.
 END                Ende der Programmierung.
                    Leitet die Benennungsroutine ein.
Programmname: xxxxxx  Wort mit maximal 6 Buchstaben
Startadresse: START   Label des Programmanfangs
```

Bem.: Zahlen können immer entweder hexadezimal oder dezimal eingegeben werden. Bei hexadezimaler Eingabe muß vor die Zahl eine spitze Klammer geschrieben werden (s.o.).

.....

Treten nach längeren Programmen Fehlermeldungen auf, so muß man das Status-Byte mit folgenden Befehlen vor dem Rücksprung löschen:

```
CLR 0            Löscht das Register 0
MOVB 0,>837C     Setzt an die Speicherstelle
                 >837C eine 0
```

Diese Befehle müssen vor LWPI GPLWS stehen.
Will man nicht zum Extended-Basic, sondern zum Titelbild zurückkehren, so gibt man am Ende ein:

```
LIMI 2
LWPI GPLWS
BLWP @>0000
```

Hilfsroutinen

Mit CALL INIT werden in den Bereich von 2000 - 24F4 mehrere Hilfsprogramme geladen, die hier kurz beschrieben werden sollen.
Sie werden allgemein mit dem Befehl BLWP @>xxxx aufgerufen. Hinter dem @ steht entweder der Programmname, falls er vorher in der Form

Name EQU Adresse

definiert worden ist, oder direkt die zugehörige Programmadresse.
Namen und Adresse können Sie der vorher aufgeführten Liste entnehmen.

Am meisten benutzt werden wohl die Hilfsroutinen zum Zugriff auf den VDP-Speicher. Die nötigen Parameter werden in den Registern übergeben.
VSBW - VDP Ram Single Byte Write
schreibt das höherwertige Byte von Register 1 an die Adresse von Register 0
Register 0: Adresse im VDP-Ram
Register 1: Im höherwertigen Byte der zu schreibende Wert
z.B.: Wenn im Register 0 eine 1000 hex. und im Register 1 eine 2345 hex. steht, dann setzt dieses Programm an die Adresse 1000 im VDP-Ram den Wert 23 hex.

.....

VMBW - VDP Ram Multiple Byte Write
Schreibt die Anzahl der Bytes, die in Register 2 vorgegeben werden, von einem Ram-Puffer, dessen Anfangsadresse in Register 1 stehen muß, in den Bereich im VDP-Ram, der mit der in Register 0 stehenden Adresse beginnt.

VSBR - VDP Ram Single Byte Read
Liest ein Byte von der VDP-Ramadresse die in Register 0 steht und schreibt es in das höherwertige Byte von Register 1.

VMBR - VDP Ram Multiple Byte Read
Liest die Anzahl der Bytes die in Register 2 stehen aus einem VDP Ram Puffer mit der Anfangsadresse aus Register 0 in einen Speicherpuffer mit der Adresse aus Register 1.

VWTR - VDP Ram Write Register
Schreibt den Wert des niederwertigen Bytes des Register 0 in das VDP Register, das durch das höherwertige Byte des Registers 0 bestimmt wird.

Zur Parameterübergabe dienen die Routinen

NUMASG	Number Assignment	NUMREF	Get Numeric Parameter
STRASG	String Assignment	STRREF	Get String Parameter

Mit ERR - Error Reporting hat man einen Zugriff auf die Fehlermeldungen des Basic-Interpreters und mit KSCAN läßt sich die Tastatur abfragen.
Die Routine XMLLNK erlaubt einen Aufruf von ROM-residenten Programmen des Betriebssystem, wie z.B. Fließkommaberechnungen. Mit GPLLNK lassen sich auch Routinen aus den graphischen Roms mit dem Assemblerprogramm verbinden. Diese Routine gibt es jedoch nur im Editor/Assembler, für dieses Assemblerpaket ist eine entsprechende Version in dem Buch 99-Special II veröffentlicht. Eine detailierte Beschreibung dieser Programme finden Sie in der angegebenen Literatur.

Beispiel

Zur Einführung sei hier ein kleines Programm und dessen Entwicklung aufgezeigt. Es soll den Buchstaben A auf der Mitte des Bildschirms darstellen. Man geht dabei wie folgt vor:
Der Rechner wird wie beschrieben mit angeschlossener Hardware gestartet. Der Assembler wird von Kassette geladen und mit RUN aufgerufen. Wenn alles bisher in Ordnung war, muß sich das Programm mit der ersten frei verfügbaren Adresse 24F4 melden. Folgende Befehle werden nun eingegeben:

GPLWS EQU >83E0	Dem Wort GPLWS (Registerbereich des Interpreters) wird der Wert >83E0 zugewiesen
VSBW EQU >2020	Dem Wort VSBW (VDP-Ram Single Byte Write) wird der Wert >2020 zugewiesen Diese Zuweisungen sind nicht notwendig. Man kann ebenso im Programm anstatt der Worte direkt die Adressen angeben. Diese Vorgehensweise hat bei längeren Programmen den Vorteil, daß man sich die Namen besser merken kann als die Adressen.
MYWS BSS 32	Ich nenne meinen Speicherbereich in dem die Register stehen MYWS. BSS reserviert dafür 32 Bytes. Man erkennt das auch daran, daß der Adresszähler (Zahl ganz links) um 32 Bytes weiter springt. Die zweite Zahl daneben zeigt vor Eingabe der Befehle den alten Wert auf dieser Speicheradresse, und nach Abschluß der Eingabezeile mit ENTER wird der vom Assembler erzeugte Maschinencode den alten Wert überschreiben. So hat man eine Kontrolle, ob das Programm richtig gearbeitet hat.
START LWPI MYWS	Den Programmanfang nenne ich START. Mit LWPI MYWS lade ich meine eigenen Arbeitsregister um nicht mit dem Basic in Konflikt zu kommen.
LI 0,>170	Dieser Befehl lädt in das Register 0 (1.Operand) die Zahl 170 hexadezimal.

Aus der Beschreibung der VSBW Routine, weiß ich, daß im Register 0 die Speicheradresse des VDP Rams stehen muß, in die das Byte geschrieben werden soll. In den ersten 300 hexadezimal (=768 dezimal) Bytes des VDP-Rams stehen die Werte, die auf dem Bildschirm erscheinen. 170 hex. müßte also auf die Mitte des Bildschirms zeigen. Beaachten Sie das Leerzeichen vor LI bei der Eingabe (diesmal kein Name oder Label gesetzt).

LI 1,>A100	Im höherwertigen Byte des Registers 1 soll der Wert des übertragenen Bytes stehen. Der ASCII-Code von A ist 65 dez. oder 41 hex. Um Zeichen auf dem Bildschirm darzustellen muß ein sogenannter "Offset" von 96 dez. bzw. 60 hex. addiert werden. Daraus ergibt sich der Wert A1.
BLWP @VSBW	Hier verzweigt das Programm zu der Hilfsroutine zum Schreiben eines Bytes in den VDP Ram.
LWPI GPLWS	Vor Ende des Programms müßen nun wieder die alten Register des Interpreters geladen werden.
RT	Rückkehr zum Extended-Basic
END	Beenden des Assembliervorgangs und Aufruf der Benennungsroutine

Nun meldet sich das Programm mit der Frage nach dem

Programmnamen:TEST und

Programmstartadresse:START

Noch ein Name (J/N)N

Ich habe nun das Programm TEST genannt. Die Startadresse meines Programms hatte den Label START. Vor diesem Label hatte ich nur Definitionen und Speicherreservierungen gemacht.

.....

Nach einer Eingabe von N wird das Programm beendet, und wir testen unser Programm direkt im Editiermodus indem wir am besten zuerst mit CALL CLEAR den Bildschirm löschen, und mit

CALL LINK("TEST)

unser Assemblerprogramm aufrufen. Es muß jetzt mitten auf dem Bildschirm ein A erscheinen.

Nach einem Einstieg in die Assemblerprogrammierung anhand dieses einfachen Beispiels, kann man durch Erweiterungen dieses Programms und Disassemblieren der mitgelieferten oder konsolinternen Programme sein Wissen vertiefen. Bei einem tieferen Eindringen in den Stoff muß man jedoch auf das Editor/Assemblerhandbuch oder ähnliche Literatur zurückgreifen. Eine Beschreibung der kompletten Möglichkeiten die dem User mit diesem Paket zur Verfügung stehen, würde den Rahmen dieser Beschreibung sprengen.

Auf der Kassette finden Sie noch eine Kurzfassung der Beschreibung dieses Assemblerpakets. Nach dem Erklärungsprogramm folgt noch ein Datensatz zu einem Demonstrationsprogramm, daß mit der Kassettenrecorder-Einleseroutine in den Speicher gebracht werden kann. Es wird mit dem Befehl CALL LINK("START") gestartet.
Bei diesem Assemblerprogramm wird der Graphikprozessor in die hochauflösende Graphik umgestellt. Es wird eine Zufallsgraphik mit räumlicher Darstellung erzeugt. Mit der Taste C kann zwischen mehrfarbig oder einfarbig umgeschaltet werden. Nach Drücken der Taste R kann man mit FCTN QUIT das Programm beenden, ohne daß es verloren geht. Mit dem Rechner kann dann normal weitergearbeitet werden, und jederzeit das Programm mit CALL LINK neu gestartet werden.

.....

ACHTUNG :

Für Assemblerprogramme steht ohne Einschränkung nur der Bereich von 24F4 bis 27xx zur Verfügung. xx hängt von der Länge der Namenstabelle ab. Der Speicher b ereich von 2800 bis 3FFF darf auf keinen Fall benutzt werden! Es kann sonst zum Programmverlust kommen.
Falls der Bereich nicht ausreicht, kann man noch auf den Bereich von 2008 bis 24F4 zurückgreifen, falls man die dort stehenden Hilfsroutinen in seinem Assemblerprogramm nicht aufruft.
Ebenfalls sind Teile des konsolinternen RAM-Bereichs von 8300 bis 83FF ansprechbar (siehe Beschreibung).
Als letzte Möglichkeit für sehr lange Programmversionen kann man auf den VDP-Ram zurückgreifen, indem man Programme oder Daten zur Bearbeitung aus dem VDP-Ram in den normalen RAM umspeichert.

Um dieses Assemblerpaket möglichst effektiv nutzen zu können, ist eine Sammlung der zur Verfügung stehenden Programme sehr sinnvoll. Wenden Sie sich diesbezüglich an

Gregor Lohmann
Mauerstraße 1a (ab 01.01.85: Bleiberger Str.54)
5100 Aachen

Literaturhinweise

Das größte Manko des geplagten TI-Users ist schon immer die mangelhafte Information gewesen. Die Lage hat sich jedoch in letzter Zeit gebessert. Daher möchte ich einige Literaturhinweise und Kontaktadressen aufzeigen, damit keine Fragen unbeantwortet bleiben.

Bei einer intensiven Auseinandersetzung mit dem Assembler geht wohl kaum ein Weg an dem Handbuch zum Editor/Assembler vorbei. Es wird zum Editor/Assemblerpaket geliefert, ist jedoch auch einzeln erhältlich. Neuerdings ist es auch in Deutsch zu haben.

Dem sehr ähnlich ist das Assemblerhandbuch für das Mini-Memory-Modul von Rainer Bernert aus Wien.

Zu empfehlen sind ebenfalls die Bücher 99 Special I und II von Alma und Johann Peschetz, TI-Learning Center 295/73830, ISBN 3-88078-043-9.

Um die Hardware besser zu verstehen sollte man den Schaltplan zur Konsole mit Erläuterungen von Texas Instruments besitzen. In Ihm stehen die Speicherbelegungen, Memory-Map Adressen und die CRU-Adressverteilung sowie die Bedeutung der verschiedenen Interrupts.

Assemblerprogramme und Fragen zum Assembler werden in letzter Zeit verstärkt in den Zeitschriften:

Computer-Praxis/Telematch
TI-Revue
TI-Journal (aus Österreich)

veröffentlicht.
Im Jahr 1983 erschienen mehrere Assemblerprogramme in der Zeitschrift Computer-Persönlich, aus der auch der mitgelieferte Assembler stammmt.

Assembler	CP Ausgabe 22 vom 19.19.83 / Ausgabe 23 vom 2.11.83
Sehr guter Disassembler	CP Ausgabe 19 vom 7.8.83
Tagget Code Editor	CP Ausgabe 20 vom 21.9.83
Permanente Farbänderung	CP Ausgabe 20 vom 21.9.83
Fliegenerator für Maschinenprogramme	CP Ausgabe 23 vom 2.11.83
40 Zeilen pro Zeile	CP Ausgabe 21 vom 5.10.83

Herr
Egon Müller
Postfach 3741
2900 Oldenburg
gibt das 99/4 Magazin in kleiner Auflage heraus, in dem auch ein Assemblerkurs abgehalten wird, in dem alles für den Anfänger verständlich beschrieben ist.

Als Kontaktadresse für Gleichgesinnte gibt es den

TI Club Aachen
Hartmut Dirks
Hans Böckler-Alle 155
Appartement 312
5100 Aachen
Tel.:0241/872205

Stand dieser Angaben ist der 1.12.1984.

Hiermit möchte ich mich bei Herrn Mag. Karl Hagenbuchner bedanken, für die Überlassung der Copyrights zum Assembler und Disassembler.